Cursus SALESFORCE

M2I - Formation - 2023

Donjon Audrey

m2iformation.fr

Module JS - Sommaire

Introduction

Rappel :

Le langage **HTML** sert à structurer les contenus d'une page Web.

Le langage **CSS** à habiller ces contenus.

Et le langage **Javascript** lui est un langage de script qui sert à modifier dynamiquement les éléments HTML/CSS de la page Web sans la recharger, en amenant une dimension d'interactivité avec l'utilisateur et le navigateur.

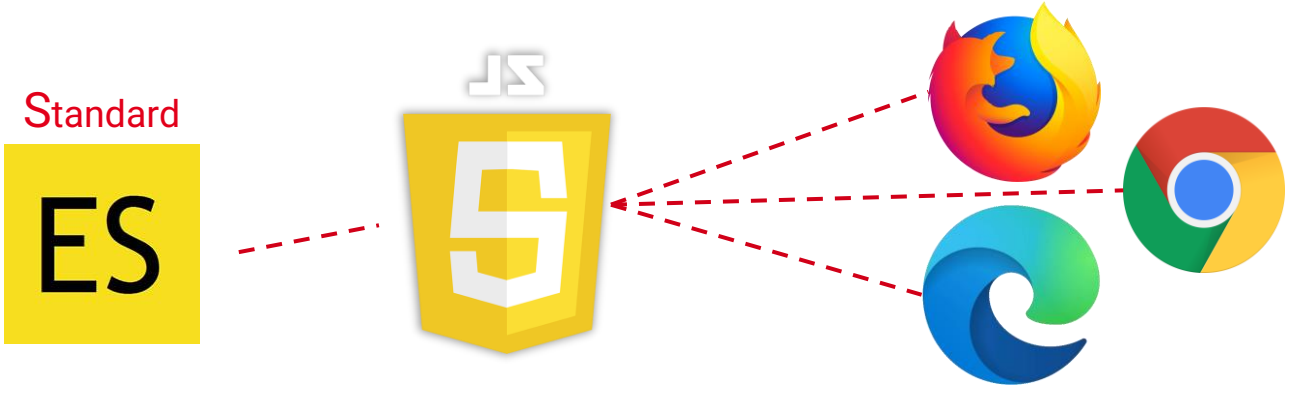
Attention : **JAVA** n'est pas **Javascript** (dit JS et non ~~Java~~ pour diminutif) :

ECMAScript

ECMAScript

Il s’agit d’ un ensemble de normes concernant les langages de programmation de type script et standardisé par Ecma International.
Il s'agit d'un standard, dont les spécifications sont mises en œuvre dans différents langages de script, notamment Javascript.

C'est ECMAScript qui définit donc comment JS doit fonctionner et tous les navigateurs Web doivent respecter ses spécifications afin que le langage fonctionne de partout pareil.



Langage de programmation

Un **langage de programmation** est un langage qui permet de décrire le comportement de la machine (de la page web pour le JS en l'occurrence) à travers ce qu'on appelle des algorithmes.

Un **algorithme** est un ensemble d'instructions qui remplit un objectif donné. (exemple : quand l'utilisateur clique sur le bouton, lui envoyer un email). Les instructions sont lues de haut en bas, l'une après l'autre (procédural).

Une **instruction** est un "ordre" donné via un langage de programmation pour effectuer une action. (exemple : Supprimer le carré rouge à l'écran)

Dans le langage JS comme dans beaucoup d'autres, une instruction doit se terminer par un point-virgule.

algorithme

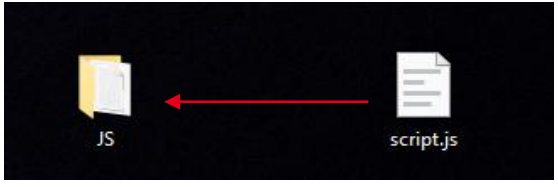
```
1  Instruction 1;
2  Instruction 2;
3  Instruction 3;
```

Sans point-virgule, le code fonctionne quand même mais il ne faut pas se reposer là-dessus : dans certains cas ça peut ne pas fonctionner donc il est considéré comme une bonne pratique de toujours mettre un point-virgule en fin d'instruction.

Le Javascript

Écrit dans 3 endroits possibles :

1 / **Dans un fichier JS** (script.js) inclut dans le code HTML (le plus utilisé car mis en cache par le navigateur)



JS

```
<!- script.js-->
alert('Fichier JS chargé !');
```

HTML

```
<!-- index.html -->
<!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset="UTF-8">
    </head>
    <body>
        <!-- contenus de ma page-->
        <script src="js/script.js"></script>
    </body>
</html>
```

2 / **Dans une balise script** directement (à éviter car pas mis en cache par le navigateur)

html

```
<!-- index.html -->
    <body>
        <script>
            alert('Code JS chargé !');
        </script>
    </body>
```

3 / **Dans la console JS du navigateur** (Ctrl + Maj + K) : Utilisé pour tester du code directement dans le navigateur

Syntaxe et documentation

- Une instruction JS se termine toujours par un point virgule **;**

- Les déclarations de variables doivent être précises - **Attention à la casse** !

  myVar et Myvar sont 2 variables différentes.

- Les commentaires s'écrivent de deux façons :

```js
// Commentaire sur une ligne

/*
    Commentaire sur
    plusieurs lignes
*/
```

- Documentations :
  - Google
  - Le MDN
  - Caniuse (pour vérifier si une fonctionnalité est supportée par certaines versions de navigateur, comme HTML et CSS)
  - Devdocs.io (documentation accessible hors-ligne, en anglais)

Quelques fonctions

Fonction console.log

La fonction "**console.log**" permet d'afficher du texte dans la console du navigateur (utilisé pour débuguer le code, faire des tests) :

```js
console.log('Cette instruction fonctionne !');
```

Fonction alert

La fonction "**alert**" permet d'afficher du texte dans la page web, dans une fenêtre modale basique du navigateur (sert à afficher un message à l'utilisateur) :

```js
alert('Cette instruction fonctionne !');
```

Fonction confirm

La fonction "**confirm**" permet de demander une confirmation à l'utilisateur (elle renverra "true" si l'utilisateur clique sur "oui", sinon "false") :

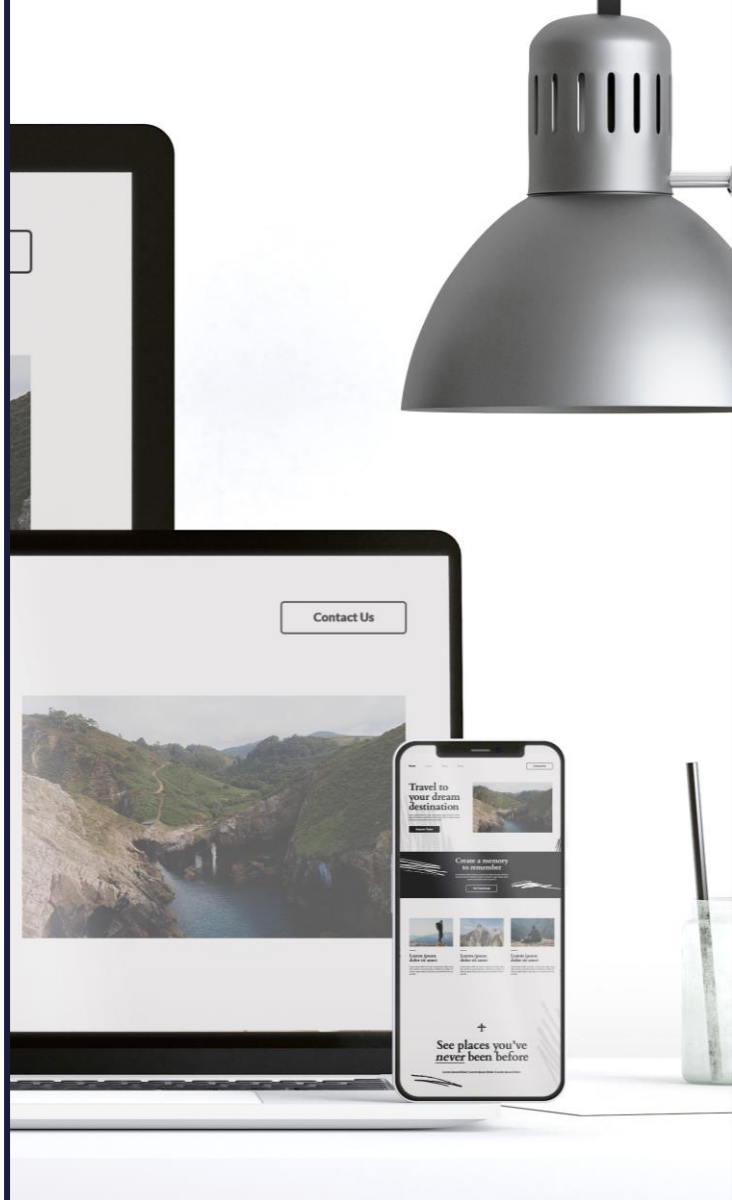
```js
let result = confirm('Êtes-vous sûr de vouloir faire cette action ?');
```

Fonction prompt

La fonction "**prompt**" permet de demander à l'utilisateur de rentrer du texte dans une boîte de dialogue (le résultat récupéré sera forcément du type "string") :

```js
let answer = prompt('Quel est votre nom ?');
```

1. Les variables



Les variables 1/2

Qu'est ce que c'est ?

Une variable est une sorte de "tiroir" dans lequel on peut stocker des informations.

Il s'agit plus exactement d'un espace mémoire réservé dans la mémoire vive (RAM) de l'ordinateur pour y stocker une information utilisée dans un algorithme (un nombre par exemple).

Une variable est composée de :

- Un nom (par exemple "age")
- Un type (par exemple "number")
- Une valeur (par exemple "36")

Création

Pour créer une nouvelle variable, on doit utiliser le mot "let" :

```JS
// Crée une nouvelle variable et lui donne une valeur
let age = 36;
```

Le type de la variable est implicite : 36 est un type "number"

Les variables 2/2

Modification

Pour modifier une variable déjà créée :

```JS
let age = 36;

// Change la valeur
age = 25;
```

Nom d'une variable

Le choix du nom d'une variable est libre mais doit respecter les règles suivantes :

- Le nom d'une variable ne peut être composé que des symboles suivants : a-z A-Z 0-9 _ $
- Le premier caractère ne doit pas être un chiffre.
- Le nom doit être toujours en anglais (convention)
- Le nom doit toujours décrire le plus explicitement son contenu (par exemple : userCity)
- les mots-clés JS sont interdits (comme "let" par exemple)
- les variables doivent être écrites en lower camel case (c'est-à-dire une majuscule à chaque mot sauf le premier. Exemple : jeSuisUneVariableDeTest)

Attention: le nom des variables est sensible à la casse (aux majuscules). Par exemple, "username" et "userName" sont deux variables différentes.

Les variables : Les types 1/2

Types de contenus

Le type d'une variable définit la nature de l'information stockée dans une variable. Liste des types communs :

string : chaîne de **caractères** (avec des **guillemets simples**, **doubles** ou **obliques** autour de la valeur).

```Js
let city = 'Paris';
let firstname = "Alice";
let lastname = `Durand`;

// Attention si vous avez besoin de stocker un guillemet dans la chaîne par exemple à bien l'échapper avec un antislash devant :
let sentence1 = 'Le chat grimpe dans l\'arbre';
let sentence2 = "Ceci est une \"citation\"";
```

number : nombres (sans rien autour de la valeur).

- La valeur "**Infinity**" est une valeur particulière de type "**number**" qui veut dire **l'infinie.**
- La valeur "**NaN**" est une valeur particulière de type "**number**" qui veut dire "**pas un nombre**".

```js
let age = 36;
let money = 125.56;
```

boolean : valeur booléenne "**vrai**" ou "**faux**" (soit **true**, soit **false**, sans rien autour de la valeur).

```js
let isAdmin = true;
let isModerator = false;
```

Les variables : Les types 2/2

Types de contenus

Le type d'une variable définit la nature de l'information stockée dans une variable. Liste des types communs :

undefined : Il s'agit de la valeur "**par défaut**" de **toutes les variables** ayant été créées sans valeur et de toutes les **fonctions sans valeur retournée**.	JS ```// Création d'une variable sans contenu let test; // Affichera "undefined" dans la console console.log(test);```
null : Signifie qu'une variable ne possède **pas de type et pas de valeur**.	JS ```let test = null;```
object	Vu plus tard dans le cours

Si besoin, il est possible de tester le type d'une variable avec « typeof » :

JS

```
let name = 'Alice';

// Affichera "string" dans la console
console.log( typeof name );
```

Les variables : opérateurs arithmétiques

Opérateurs arithmétiques

Comme tous les langages de programmation, JS gère les calculs mathématiques. On peut ainsi utiliser les opérateurs arithmétiques classiques :

JS

```js
let number = 5;

// Addition
console.log( number + 5 );  // Affichera 10

// Soustraction
console.log( number - 20 );  // Affichera -15

// Multiplication
console.log( number * 10 );  // Affichera 50

// Division
console.log( number / 2 );  // Affichera 2.5

// Modulo (reste de la division)
console.log( number % 2 );  // Affichera 1
```

Comme dans les mathématiques, il faut faire attention aux priorités de calculs :

JS

```js
console.log( 5 + 2 * 4 );   // Affichera bien 13 (et non 28, car la multiplication est prioritaire sur l'addition)

console.log( (5 + 2) * 4 ); // On peut changer la priorité grâce aux parenthèses (affichera 28)
```

Les variables : concaténation et opérateurs d'affectation 1/2

Concaténation

La concaténation est l'action de "coller" deux chaînes de texte ensemble :

JS

```js
let name = 'Alice';

// Affichera "Bonjour Alice !"
console.log('Bonjour ' + name + ' !');

// Possible de faire la même chose différemment UNIQUEMENT avec les guillemets obliques `` autour :
console.log(`Bonjour ${name} !`);
```

Opérateurs d'affectation

Les opérateurs d'affectation permettent d'assigner une nouvelle valeur à une variable. Le plus simple étant le signe "=". Il en existe d'autres permettant de combiner plusieurs opérateurs :

JS

```js
let number;

// Affectation simple de valeur
number = 25;

// Double action : calcul de la valeur actuelle de la variable + 5 ET affectation de la nouvelle valeur dans la variable (number vaux maintenant 30, résultat de 25 + 5)
number += 5;    // 30

// Double action : calcul de la valeur actuelle de la variable - 10 ET affectation de la nouvelle valeur dans la variable (number vaux maintenant 20, résultat de 30 - 10)
number -= 10;   // 20
```

Les variables : concaténation et opérateurs d'affectation 2/2

JS

```
-- Suite –

Let number;

// Double action : calcul de la valeur actuelle de la variable * 3 ET
affectation de la nouvelle valeur dans la variable (number vaux maintenant 60,
résultat de 20 * 3)
number *= 3;    // 60

// Double action : calcul de la valeur actuelle de la variable / 4 ET
affectation de la nouvelle valeur dans la variable (number vaux maintenant 15,
résultat de 60 / 4)
number /= 4;    // 15

// Double action : calcul de la valeur actuelle de la variable % 4 ET
affectation de la nouvelle valeur dans la variable (number vaux maintenant 3,
résultat du modulo de 15 par 4)
number %= 4;    // 3
```

Les variables : Incrémentation et décrémentation

L'incrémentation est l'action de changer la valeur d'une variable pour lui ajouter 1 (passer de 22 à 23 par exemple). Il existe la même chose dans l'autre sens, c'est-à-dire enlever 1 : la décrémentation.

L'incrémentation	JS ```let number = 20; // Incrémentation number++; // Affichera "21" console.log(number);```
la décrémentation	JS ```let number = 10; // Décrémentation number--; // Affichera "9" console.log(number);```

Les variables : Changer le type d'une variable 1/2

Parfois une variable peut être d'un type qui ne nous convient pas. Il est possible grâce à certaines fonctions de "transtyper" une variable, c'est-à-dire changer son type tout en préservant autant que possible sa valeur.

Dans l'exemple suivant, on récupère l'âge d'un utilisateur via la fonction "prompt" pour effectuer dessus un petit calcul ->

Le problème ici c'est que prompt renvoi toujours la valeur récupérée sous la forme d'une "string". Quand on essaie donc d'ajouter 5 à la valeur "25", il en résulte une concaténation ("25" + 5 = 255) au lieu d'une addition, car pour JS "25" est une chaîne de texte.

JS

```js
let userAge = prompt('Quel est votre âge ?');

userAge += 5;

// Ici si l'utilisateur entre "25", le résultat sera : "Dans 5 ans, vous aurez 255 ans !"
alert('Dans 5 ans, vous aurez ' + userAge + ' ans !');
```

Pour résoudre le problème, il faudrait donc que l'âge récupéré soit "converti" en type "number" au lieu d'être une "string" ->

JS

```js
let userAge = prompt('Quel est votre âge ?');

// On "force" la variable à être un entier de type "number"
userAge = parseInt( userAge );

userAge += 5;

// Ici si l'utilisateur entre "25", le résultat sera : "Dans 5 ans, vous aurez 30 ans !"
alert('Dans 5 ans, vous aurez ' + userAge + ' ans !');
```

Les variables : Changer le type d'une variable 2/2

Quelques méthodes pour changer de type :

JS

```
// Force à être de type "number" et entier
console.log( parseInt("25.36") );   // Affichera 25

// Force à être de type "number"
console.log( parseFloat("25.36") );   // Affichera 25.36

// Attention, si parseInt et parseFloat ne contiennent pas de nombre valide,
le résultat sera NaN
console.log( parseInt("Je suis pas un nombre !") );   // Affichera NaN

// Force à être de type "string"
console.log( String(56) );  // Affichera "56" (en chaîne de texte)
```

Les variables : Les constantes

Les constantes sont comme des variables (informations stockées dans la mémoire avec un type et une valeur) mais contrairement à ces dernières, une fois créées elles ne peuvent plus changer de valeur :

JS

```js
// Création d'une constante
const name = 'Alice';

// Affiche "Alice"
console.log( name );

// Erreur, une constante ne peut pas être modifiée
name = 'Bob';
```

L'utilité des constantes réside dans le fait que leur valeur est inaltérable : c'est une garantie supplémentaire que cette valeur ne pourra pas être changée dans le script.

Exercice 01

Votre script doit :

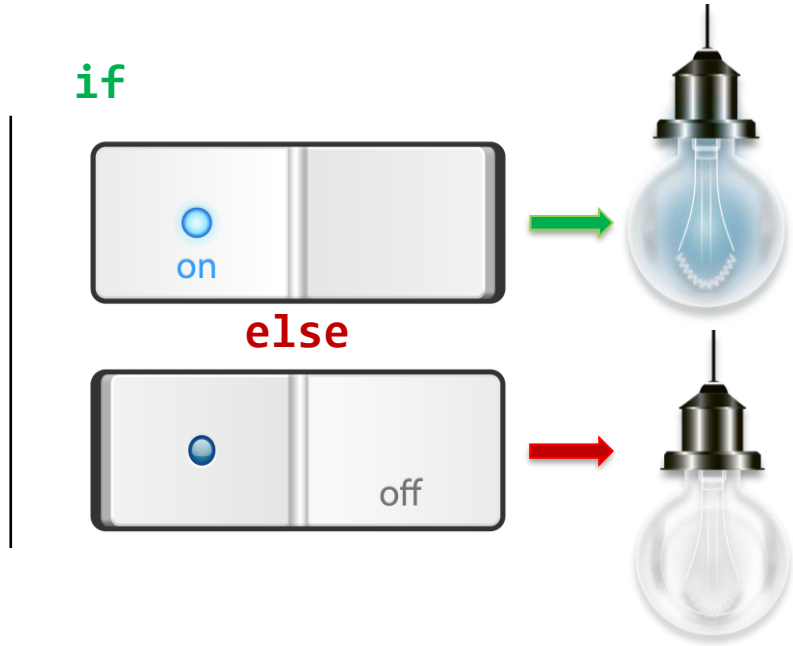
1. Avoir 2 variables, implémenter avec des valeurs de type number
2. Avoir une constante qui est égal à la somme des 2 variables
3. Afficher un message d'alerte « Bienvenue sur la page de formulaire »
4. Afficher dans la console du navigateur le contenu des variables
5. Afficher dans la console « Le résultat est » + le résultat du calcul

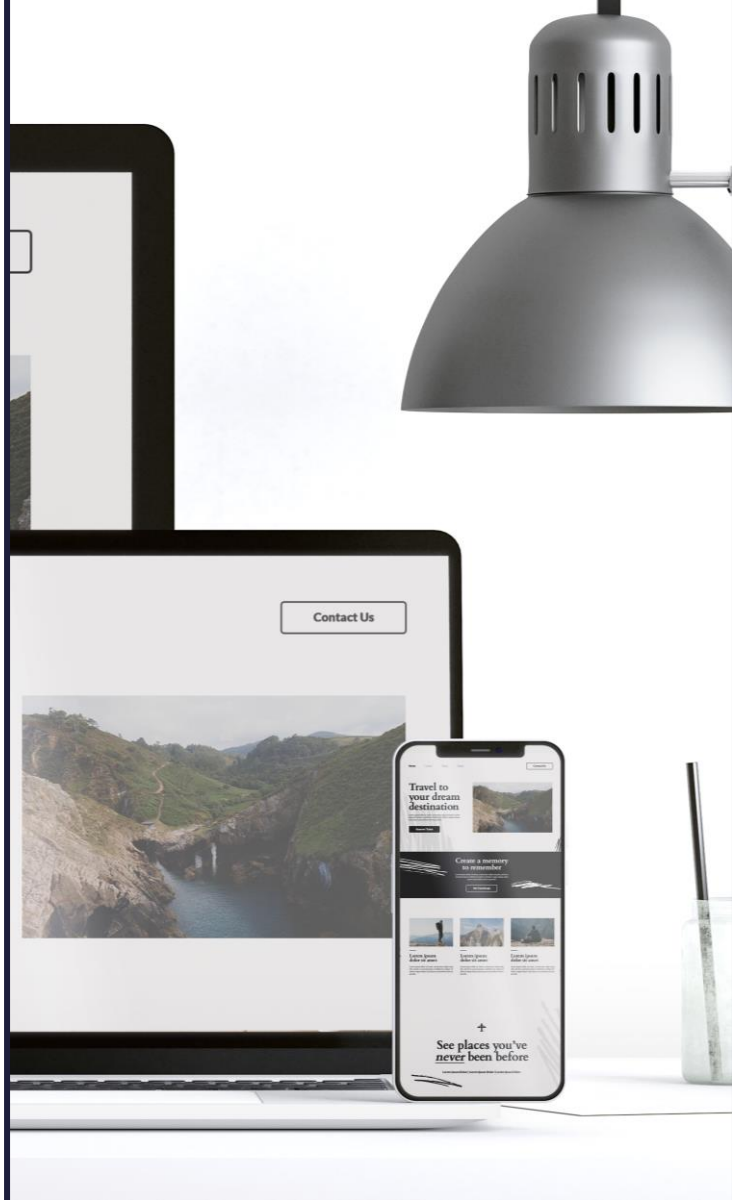
Exercice 02

Énoncé :

1. Écrire un programme JS qui demande d'abord à l'utilisateur d'entrer, à partir du clavier la distance parcourue en m (mètre), puis le temps mis à parcourir cette distance en s (secondes).
2. Calculer la vitesse selon la formule suivante : vitesse = distance parcourue/temps.
3. Afficher le résultat dans le format suivant : 345 **m/s**. Utiliser '**alert**' et '**console.log**'.

2. Les conditions





Les conditions

Une condition est un bloc d'instructions qui ne sera exécuté que si son test est vrai.

Le "test" d'une condition est matérialisé sous la forme d'une expression qui sera évaluée au final comme étant vrai (true) ou fausse (false) :

```js
// l'expression 5 plus petit que 10 est vrai, donc la condition sera bien lue
if(5 < 10){
    // Sera bien lu
    alert('Bonjour !');
}

let age = 5;
// l'expression "age" plus grand ou égal que 18 est fausse car la variable contient "5", donc tout le bloc de la condition sera ignoré
if(age >= 18){
    // Ne sera pas lu
    alert('Bienvenue sur notre site !');
}
```

Une condition est une structure de code et non une instruction à proprement parler, il ne faut donc pas mettre de point-virgule à la fin d'un if ou d'une accolade.

Opérateurs de comparaison 1/3

Pour construire les conditions, il faut utiliser des opérateurs de comparaison :

Opérateur « valeur plus petite que » :	JS ```let test = 50; // Condition vraie car 50 est bien plus petit que 60 if(test < 60){ // ... }```
Opérateur « valeur plus petite ou égale à »:	JS ```let test = 50; // Condition vraie car 50 est bien plus petit ou égal à 80 if(test <= 80){ // ... }```
Opérateur « valeur plus grande que » :	JS ```let test = 50; // Condition vraie car 50 est bien plus grand que 30 if(test > 30){ // ... }```

Opérateurs de comparaison 2/3

Opérateur «valeur plus grande ou égale à» :

```js
let test = 50;

// Condition vraie car 50 est bien plus grand ou égal à 45
if( test >= 45 ){
    // ...
}
```

Opérateur « valeur égale à »

(vérifie uniquement la valeur, pas le type !) :

```js
let test = 50;
// Condition vraie car 50 est bien égal à 50
if( test == 50 ){
    // ...
}
// Condition vraie aussi car 50 et "50" ont bien la même valeur
(même si le type est différent !)
if( test == "50" ){
    // ...
}
```

Opérateur « valeur différente de » :

```js
let test = 50;

// Condition vraie car 50 est bien différent de 0
if( test != 0 ){
    // ...
}
```

Opérateurs de comparaison 3/3

Opérateur "valeur ET type égaux à"

(vérifie la valeur mais aussi le type cette fois !) :

```JS
let test = 50;
// Condition vraie car 50 est bien égal à 50 et de même type (number en l'occurrence)
if( test === 50 ){
    // ...
}

// Condition fausse car 50 et "50" ont bien la même valeur mais ils sont d'un type différent ! ("50" = string alors que 50 = number)
if( test === "50" ){
    // ...
}
```

Opérateur "valeur OU type différent de"

(Il suffit que la valeur ou le type soit différent pour que la condition soit vraie) :

```JS
let test = 50;
// Condition vraie car 50 et "50" ont un type différent
if( test !== "50" ){
    // ...
}

// Condition vraie car 50 et 52 ont une valeur différente
if( test !== 52 ){
    // ...
}
```

Opérateurs logiques

Les opérateurs logiques servent à combiner plusieurs "tests" dans une condition (sauf pour "NO") :

Opérateur logique "&&" (AND) : Pour que la condition soit vraie, il faut que les deux tests soient vrais.

```JS
let age = 25;

// Condition vrai car les 2 tests sont satisfaits : 25 est bien plus grand ou égal à 20 et il est également plus petit que 30
if(age >= 20 && age < 30){

    alert('Bienvenue sur ce site réservé aux personnes ayant la vingtaine !');
}
```

Opérateur logique "||" (OU) : Pour que la condition soit vraie, il faut qu'au moins un des deux tests soit vrai.

```JS
let temperature = 60;

// Condition vrai car "temperature" est plus grande que 40
if(temperature < 0 || temperature > 40){

    alert('La température est dangereuse !');
}
```

Opérateur logique "!" (NO) : Inverse le sens d'un booléen. Cet opérateur est très utilisé pour inverser le sens d'une fonction ou d'une variable.

```JS
// Utilisateur non autorisé (c'est un exemple purement démonstratif)
let authorizedUser = false;

// Dans cette situation, on souhaite entrer dans la condition si la variable contient false. On peut donc inverser le sens avec l'opérateur "!"
if(!authorizedUser){

    alert('Vous n\'êtes pas un utilisateur autorisé !');

}
```

Structures conditionnelles 1/3

Structure else

Il est possible d'adjoindre avec "else" un autre bloc qui sera exécuté dans le cas où une condition est fausse :

```JS
let temperature = 30;

// Condition fausse car "temperature" n'est pas plus petite que 0, ni plus grande que 40 : c'est donc le bloc "else" qui sera exécuté
if(temperature < 0 || temperature > 40){

    alert('La température est dangereuse !');
} else {

    alert('La température est bonne !');
}
```

Structures conditionnelles 2/3

Structure else if

Dans une structure if..else il n'y a que deux possibilités : soit c'est vrai, soit c'est faux. Grâce à "else if", on peut imbriquer d'autres conditions dans la même structure conditionnelle :

```JS
let city = 'Lyon’;
if(city == 'Paris'){
    // Faux, passe au if suivant
    alert('Bonjour à toi le parisien !');

} else if(city == 'Lyon'){
    // vrai
    alert('Bonjour à toi le lyonnais !');

} else {
    // Pas lu
    alert('Désolé je ne connais pas ta ville');
}
```

Il peut y avoir autant de "else if" que l'on souhaite en mettre, cependant il faut bien se rappeler que cet ensemble reste une seule structure conditionnelle et donc qu'une seule sortie est possible :

```JS
let test = 50;
if(test > 40){
    // Sera bien lu
    alert('test est plus grand que 40 !');

} else if(test > 45){
    // Même si la condition est vraie, elle ne sera pas lue car la première était vraie aussi et qu'une seule sortie est possible dans une même structure conditionnelle !
    alert('test est plus grand que 45 !');
}
```

Structures conditionnelles 3/3

Conditions ternaires

Syntaxe :

condition ? exprSiVrai : exprSiFaux

Une condition ternaire est une manière très raccourcie d'écrire une condition pour affecter une valeur. Exemple :

En savoir plus sur les ternaires :

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Operators/Conditional_Operator

JS

```
let temperature = 30;

// Code classique
if(temperature < 15){

    alert('Il fait froid !');
} else {

    alert('Il fait chaud !');
}


// Même chose avec une ternaire
alert( (temperature < 15) ? 'Il fait froid !' : 'Il fait chaud !' );
```

Exercice 03

Énoncé :

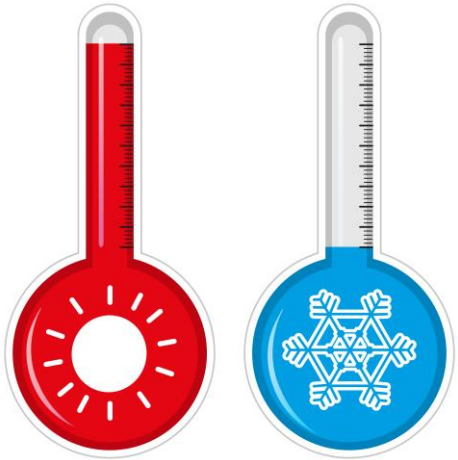
Écrire un programme Javascript qui demande l'email et le mot de passe de l'utilisateur, qu'on aura au préalable défini dans notre script.

Vérifier la condition suivante :

- Si l'email et le mot de passe correspondent aux valeurs définies, le message d'alerte devra afficher « Bienvenue dans votre espace client »
- Sinon, le message d'alerte devra afficher « Identifiants incorrects »

Les boucles

```
Let temp = 0
```

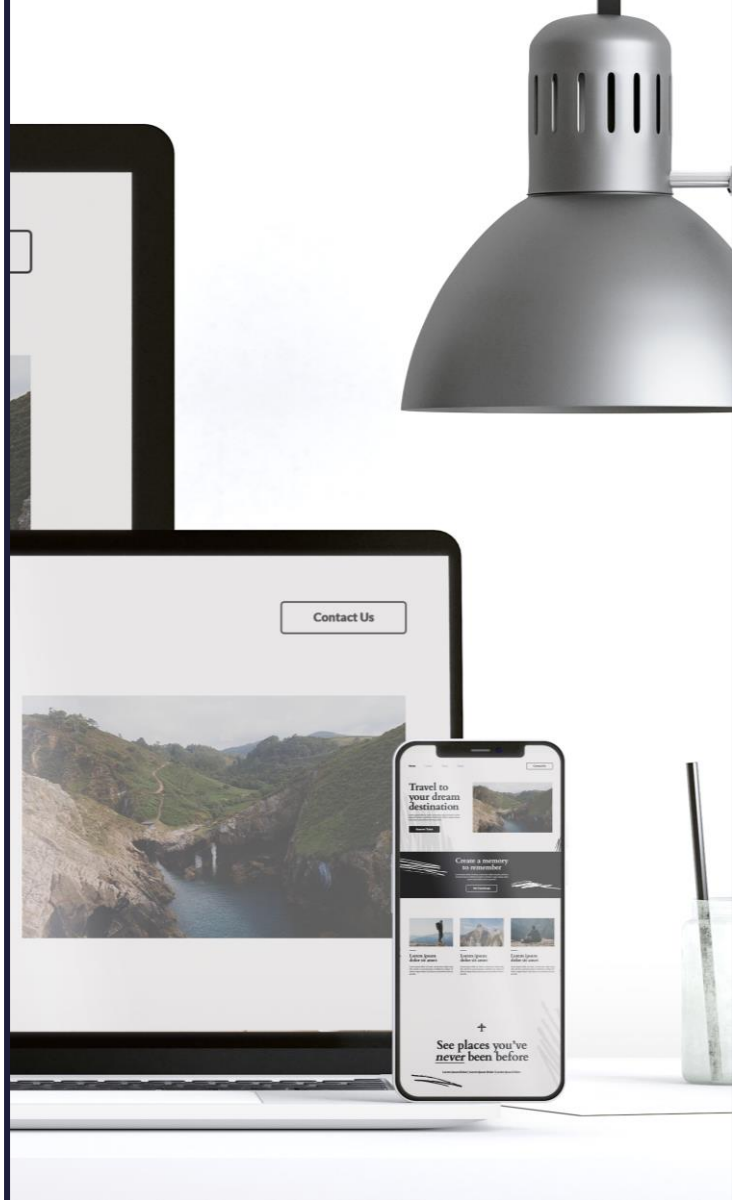


```
While(temp => 25)
```

Afficher *image 1*
Avec le *message* :
« il fait chaud aujourd'hui »

Enlever *image 2* afficher par défaut avec le *message*
« il fait froid aujourd'hui »

```
temp++
```



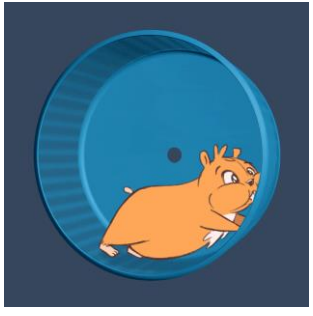
Les boucles

Qu’est ce que c’est ?

En programmation, une boucle est une structure permettant de répéter des instructions plusieurs fois sans être obligé de les écrire plusieurs fois.

Chaque boucle est dotée d'un test (comme les conditions), et tant que cette condition est vraie la boucle recommencera.

- **Attention** : si on met un test toujours vrai, ça produira une boucle infinie (qui peut planter le navigateur !)
- Si la condition de la boucle est fausse dès le départ, la boucle sera tout simplement ignorée.



Mon hamster courra tant que son énergie ne sera pas en dessous de 40%

Boucle While

La boucle while est la boucle la plus classique, qui s'exécutera tant que sa condition sera vraie.

Pour éviter de faire une boucle infinie, on utilise souvent une variable qui servira à compter combien de fois la boucle s'est déjà exécutée. Par convention cette variable s'appelle "i" (car c'est un itérateur qui s'incrémente de 1 à chaque tour de boucle) :

JS

```
// Itérateur à 0 au début
let i = 0;

// La boucle s'exécutera tant que i sera plus petit que 10
while(i < 10){
    alert('Je suis une boucle !');

    // Très important pour augmenter le compteur, sinon boucle infinie !
    i++;
}
```

Boucle for et do while

La boucle for est une boucle avec un itérateur intégré directement dans sa structure (même chose que while sinon) :

```js
// La boucle s'exécutera tant que i sera plus petit que 10
for(let i = 0; i < 10; i++){

    alert('Je suis une boucle !');

}
```

La boucle do while est une boucle while qui s'exécutera au moins une fois même si sa condition est fausse dès le départ :

Contrairement aux autres structures, il faut mettre un point-virgule à la fin d'une boucle do while !

```js
// Cette alerte sera lue quand même une fois même si la condition est fausse
do{

    alert('bonjour');

} while(false);
```

Break

Si dans une boucle pour une raison ou une autre vous souhaitez arrêter la boucle avant la fin, vous pouvez avec "break", ce qui permettra de sortir immédiatement de la boucle :

```js
// Boucle infinie (mais contrôlée car prompt met le code en pause à chaque tour, donc pas de risque de plantage !)
while( true ){

    // On demande le mot de passe à l'utilisateur
    let userAttempt = prompt('Quel est le mot de passe ?');

    // Si le mot de passe entré est "azerty", c'est bon
    if(userAttempt == 'azerty'){

        alert('C\'est le bon mot de passe !');

        // Stop la boucle pour qu'elle ne continue pas
        break;
    }

    alert('Mauvais mot de passe, veuillez ré-essayer.');

}
```

Exercice 04

Énoncé :

Améliorez le programme Javascript fait lors de **l'exercice 03** en faisant bouclé la demande du mail et du mot de passe tant que ceux-ci ne seront pas correct.

Exercice 05

Énoncé :

Écrire un programme Javascript qui demande à l'utilisateur un nombre compris entre 1 et 3.

- Tant que l'utilisateur n'aura pas retourné un nombre entre 1 et 3 (pensez à utiliser une condition pour vérifier le nombre), il faudra lui re demander jusqu'à ce que la réponse convienne.

Exercice 06

Énoncé :

Écrire un programme Javascript qui demande à l'utilisateur de rentrer un nombre.

Ensuite ce nombre devra être utilisé pour affiché sa table de multiplication et devra être affiché dans la console comme suit :

Exemple avec le nombre 7 rentré :

7 x 1 = 7

7 x 2 = 14

7 x 3 = 21

....

7 x 10 = 70

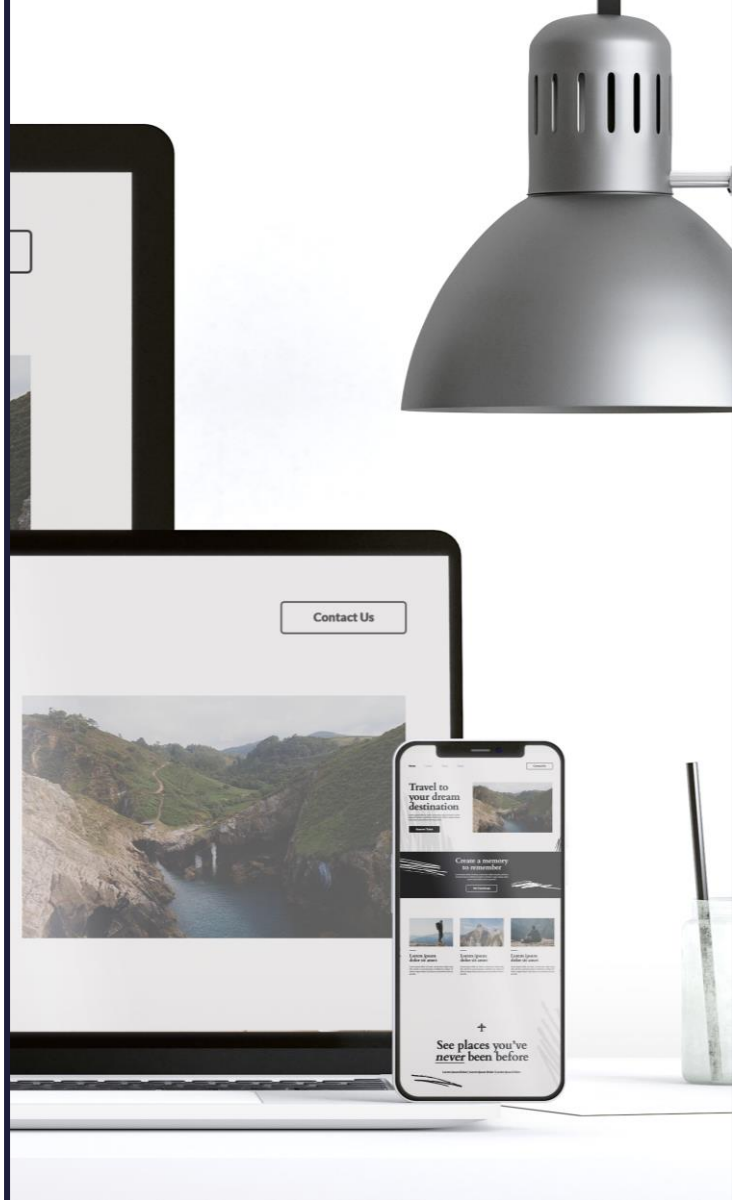
Les fonctions

```
function drink(){
alert('Le chat boit')
}
```


```
drink();
```

```
function eat(){
alert('Le chat mange')
}
```

```
eat();
```



Les Fonctions

Qu'est ce que c'est ?

Une fonction est un regroupement d'instructions qui produit un résultat. Elle permet de faire appel à ces instructions sans être obligé de les retaper à chaque fois.

Pour appeler une fonction, on utilise son nom suivi d'une paire de parenthèse :

```JS
// Invocation de la fonction alert
alert();
```

- Il existe déjà de base dans JS plein de fonctions natives comme "alert()", "confirm()" ou "parseInt()" par exemple.

Certaines fonctions peuvent accepter des valeurs entre leurs parenthèses : on appelle ces valeurs des arguments :

```JS
// Invocation de la fonction alert avec une chaîne de texte en argument
alert('Bonjour !');
```

Créer une fonction

Un développeur aura toujours besoin de créer des fonctions sur mesure pour remplir des tâches que les fonctions natives de Javascript ne savent pas faire. Pour créer une fonction il faut la déclarer :

JS

```
// Déclaration de la nouvelle fonction
function sayHello(){

    // Ici le corps de la fonction, c'est-à-dire le code qui la compose
    alert('Bonjour !');

}

// On peut maintenant utiliser cette nouvelle fonction en l'invoquant
sayHello();
```

- Le nom d'une fonction doit être écrit en lower camel case (c'est-à-dire une majuscule à chaque mot sauf le premier. Exemple : encoreUneFonction())

Paramètres de fonction

Les paramètres de fonction vont permettre de faire rentrer des valeurs dans la fonction au moment où cette dernière est appelée.

```js
// La fonction pourra récupérer un prénom en argument grâce au paramètre "name"
function sayHello(name){

    // On peut utiliser la variable "name" dans la fonction comme on le souhaite
    alert('Bonjour ' + name + ' !');

}

// Dire bonjour à Alice
sayHello('Alice');

// Dire bonjour à Bob
sayHello('Bob');
```

Valeur par défaut à un paramètre

Un paramètre de fonction peut avoir une valeur par défaut si vous le souhaitez :

JS

```js
// Le paramètre "name" contiendra "John Doe" si jamais ce dernier n'est pas rempli lors de l'appel de la fonction
function sayHello(name = 'John Doe'){

    alert('Bonjour ' + name + ' !');

}

// Dire bonjour à Alice
sayHello('Alice');

// Dire bonjour à John Doe (car il n'y a pas de paramètre donc ce dernier aura sa valeur par défaut)
sayHello();
```

Retourner un résultat

Si une fonction produit un résultat qui doit être utilisé dans le code, il faut que cette dernière retourne ce résultat avec le mot-clé 'return'

JS

```
// Fonction qui doit calculer et retourner le résultat du triple du nombre entré en paramètre
function triple(number){
    return number * 3;
}

// Grâce au return de la fonction, le résultat (120) pourra être récupéré et stocké dans la variable result
let result = triple(40);
```

- En général une fonction va soit faire une action et ne rien retourner (fonction alert() par exemple), soit calculer quelque chose et retourner le résultat (comme parseInt() par exemple)
- L'instruction "return" stop le code de la fonction. Si du code est placé après il ne sera jamais lu (sauf si le return est dans une condition).

Portée des variables

La portée des variables est un concept important qui définit où sont accessibles les variables que l'on crée dans notre code. Par rapport aux fonctions, il existe deux sortes de variables :

Les variables locales : ce sont les variables créées directement dans une fonction et qui n'existent que dans cette fonction. Les variables locales d'une fonction n'existent que pour elle :

```JS
function test(){

    let firstname = 'Alice';

    // Fonctionne car la variable "firstname" existe dans le corps de la fonction (elle est locale à la fonction)
    alert(firstname);
}

// Erreur variable introuvable : ayant été créée dans une fonction, elle n'existe que dans cette fonction
alert(firstname);
```

Les variables globales : ce sont les variables créées normalement, en dehors d'une fonction et qui existent de partout.

ATTENTION : Utiliser une variable globale directement dans une fonction est possible (comme vu ici) mais doit être évité le plus possible ! En effet le principe de base d'une fonction est d'être un outil fonctionnel en étant le plus autonome possible. Une fonction qui utilise des variables globales est dépendante de son contexte d'utilisation (c'est-à-dire que la fonction sera dépendante de l'extérieur pour fonctionner correctement).

```JS
let firstname = 'Alice';

function test(){

    // Fonctionne car la variable "firstname" a été créée en dehors de la fonction et est accessible aussi dans les fonctions: c'est une variable globale
    alert(firstname);
}

// Fonctionne normalement
alert(firstname);
```

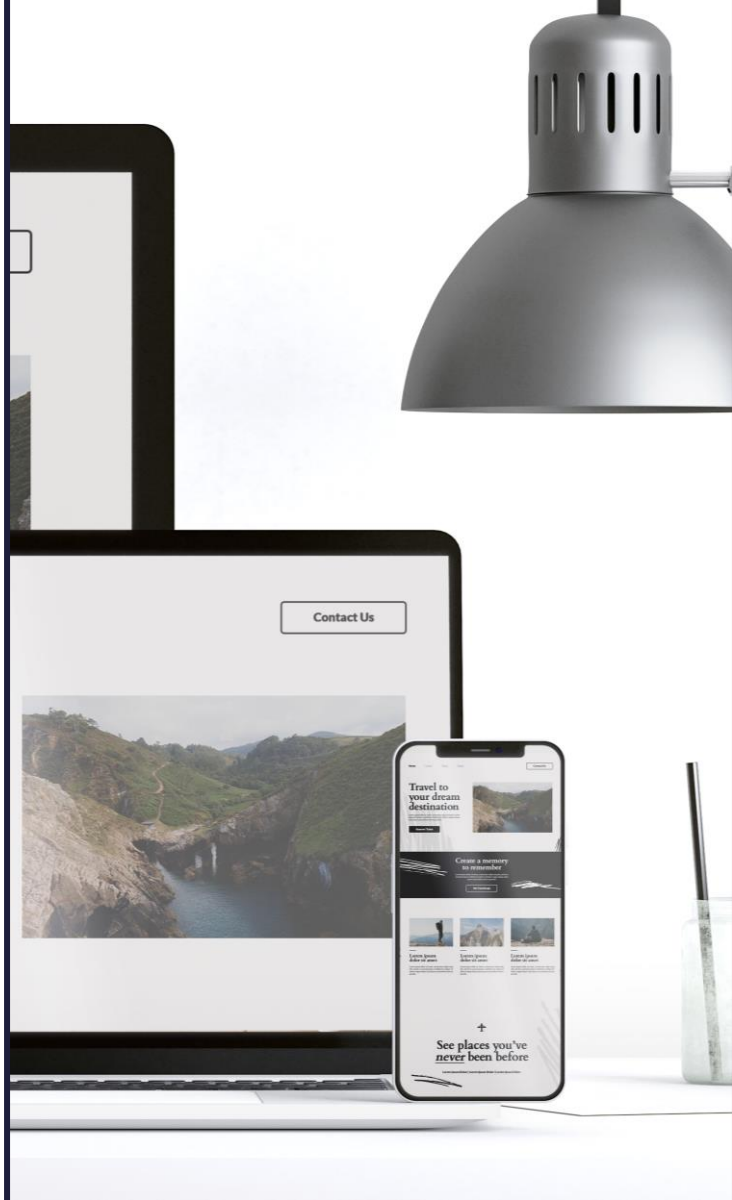
Exercice 07

Énoncé :

Écrire une fonction Javascript qui prend en paramètre 2 nombres et ensuite retourner leur somme.

Les arrays

```
let chocolates = ['white chocolate’,'dark chocolate’,
'milk chocolate'];
```



Les arrays

Qu'est ce que c'est ?

Un array (tableau de données en français) est un objet permettant de stocker plusieurs valeurs dans une seule variable (comme une sorte de "sac à variables") :

```js
// Création d'un array contenant des animaux
let animals = ['chat', 'chien', 'loutre', 'hérisson'];
```

Pour accéder à un élément du tableau (pour l'afficher par exemple), il faut l'appeler via son index, c'est-à-dire sa position dans l'array en partant du zéro pour le premier :

```js
// Création d'un array contenant des animaux
let animals = ['chat', 'chien', 'loutre', 'hérisson'];

// Affichera "loutre" (chat = 0, chien = 1, loutre = 2, hérisson = 3)
alert( animals[2] );
```

Attention : le compte des éléments dans un array commence à partir de 0 !

Quelques fonctions utiles sur les arrays 1/2

unshift : Ajoute un élément au début du tableau

```js
let animals = ['chat', 'chien', 'loutre', 'hérisson'];

animals.unshift('lapin');

// Affichera "lapin" (lapin = 0, chat = 1, chien = 2, loutre = 3, hérisson = 4)
alert( animals[0] );
```

push : Ajoute un élément à la fin du tableau

```js
let animals = ['chat', 'chien', 'loutre', 'hérisson'];

animals.push('lapin');

// Affichera "lapin" (chat = 0, chien = 1, loutre = 2, hérisson = 3, lapin = 4)
alert( animals[4] );
```

shift : Supprime le premier élément du tableau

```js
let animals = ['chat', 'chien', 'loutre', 'hérisson'];
// Supprime chat
animals.shift();

// Affichera "chien" (chien = 0, loutre = 1, hérisson = 2)
alert( animals[0] );
```

Quelques fonctions utiles sur les arrays 2/2

pop : Supprime le dernier élément du tableau

JS

```js
let animals = ['chat', 'chien', 'loutre', 'hérisson'];
// Supprime hérisson
animals.pop();

// Affichera "undefined" car il n'existe plus d'élément avec l'index 3 (chat = 0, chien = 1, loutre = 2)
alert( animals[3] );
```

Récupérer et parcourir un array

Pour récupérer la taille d'un array (nombre d'éléments dedans), on utilise "**length**" :

```js
let animals = ['chat','chien','loutre','hérisson'];

// Affichera "4"
alert(animals.length);
```

Il est très courant d'avoir besoin de parcourir chaque élément d'un tableau. Pour automatiser ça on peut utiliser les boucles, dont une qui a été créée spécialement à cet effet : la **boucle forEach**

```js
let animals = ['chat','chien','loutre','hérisson'];
// Boucle forEach pour parcourir tous les éléments du tableau "animals", un à un (à chaque tour de boucle) :
animals.forEach((animal) => {
     // le paramètre "animal" est une variable qui contient un animal différent à chaque tour de boucle
    alert (animal);
});
```

Pour résumer, la **boucle forEach** permet de faire des actions sur tous les éléments d'un tableau. Chacun de ces éléments est extrait puis mis à disposition dans le paramètre de la fonction de retour entre les parenthèses de la boucle ("animal" dans l'exemple, le nom est au choix du développeur !).

Array multidimentionnels

Un array multidimensionnel est simplement un array contenant d'autres arrays. Exemple :

JS

```
// Array bidimensionnel contenant 3 arrays

let animals = [
    ['Chat','Européen’],
    ['Chien','Husky'],
    ['Loutre','d\'Europe'],
];

// Pour accéder à une information précise d'un élément, on doit mettre une
paire de parenthèse pour chaque profondeur de tableau :

// Affichera "Chien" (array animals => deuxième élément [1] => premier
élément [0])
alert (animals[1][0]);
```

Exercice 08

Énoncé :

Écrire **un programme Javascript** qui :

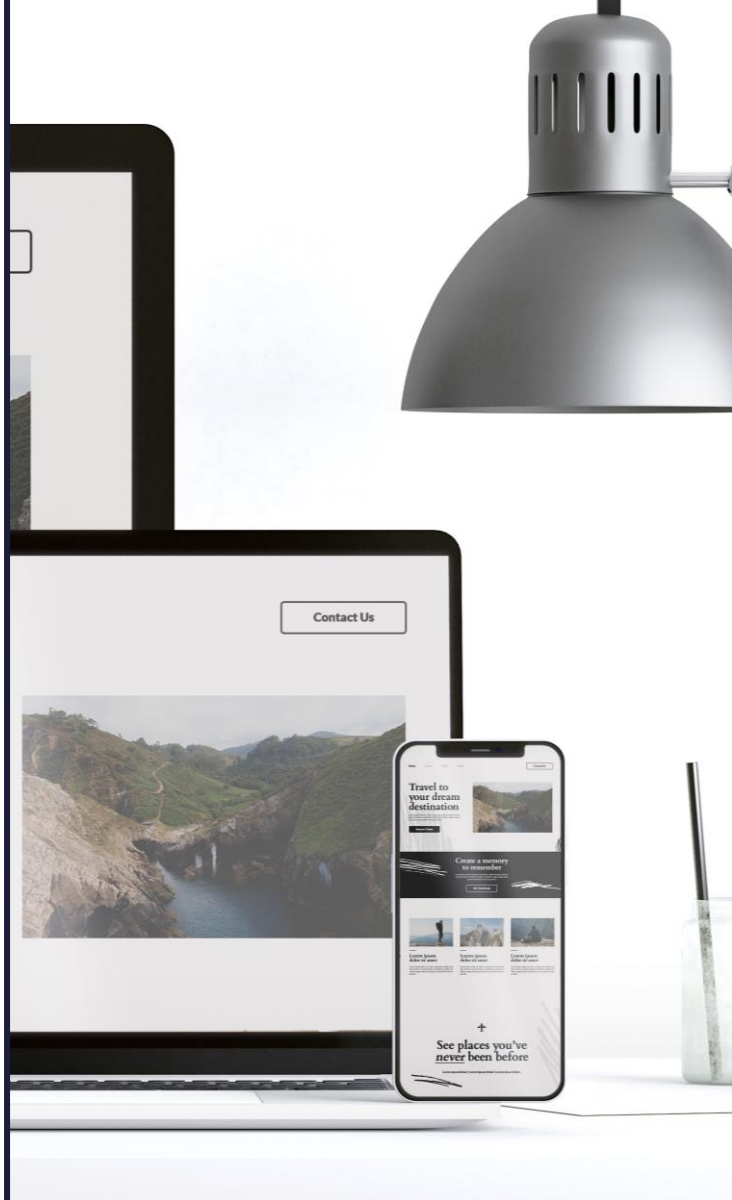
1. Va demander confirmation à l'utilisateur si il souhaite ajouter un nouveau prénom puis va demander « Quel prénom souhaitez-vous ajouter à la liste ? »
2. Après avoir rentrer le prénom, il lui sera re demandé si il souhaite ajouter un nouveau prénom
3. Tant que l'utilisateur entrera un prénom à ajouter, rajouter ce prénom dans un array et l'afficher dans la console

Les objets

```
// Un objet matérialisant un mage
let wizard = {

    // Un attribut de l'objet (variable)
    element: 'crystal',

    // Une méthode de l'objet (fonction)
    heal: function(){
        alert('Le mage lance un sort de soin !');
    },
};
```



Les objets 1/2

Qu'est ce que c'est ?

Les objets sont des éléments de programmation qui possède des variables (appelées des **attributs**) et des fonctions (appelées des méthodes). Contrairement aux variables et aux fonctions habituelles, ces dernières n'existent qu'à l'intérieur de leur objet.

```js
// Un objet
let car = {

    // Un attribut de l'objet (variable)
    color: 'red',

    // Une méthode de l'objet (fonction)
    start: function(){
        alert('La voiture démarre');
    },

};
```

Objet : Voiture

Une méthode de l'objet :
La voiture démarre

Un attribut de l'objet :
couleur : rouge

Les objets servent aussi à créer des **listes d'éléments** (comme les arrays) mais avec des **index personnalisés** (au lieu de 0, 1, 2, etc...) :

```js
// Objet matérialisant un bonbon
let candy = {
    type: 'nounours',
    color: 'red',
};

// Deux moyens pour accéder aux éléments d'un objet :
// Affichera "nounours"
alert( candy.type );

// Affichera "nounours" aussi
alert( candy['type'] );
```

Les objets 2/2

Les objets permettent de créer des structures autonomes qui centralisent tous les éléments nécessaires à leur fonctionnement. On pourrait par exemple imaginer un objet pour créer un lecteur vidéo et y intégrer dedans toutes ses propriétés (dimensions, fichier lu, etc...) ainsi que toutes ses fonctionnalités (mettre en pause, monter le son, etc...)

JS

```js
// Création d'un objet pour gérer un lecteur vidéo (exemple simple, pas un vrai !)
const videoPlayer = {

    // Attributs (variables) de l'objet
    height: 200,
    width: 500,

    // Méthodes (fonctions) de l'objet
    play: function(){
        alert('Vidéo lancée !');
    },
    pause: function(){
        alert('Vidéo en pause !');
    },

};

// Accès aux éléments de l'objet :

// Affichera "200"
alert( videoPlayer.height );

// Exécutera la fonction play dans l'objet "videoPlayer"
videoPlayer.play();
```

Les objets : This

Si une méthode (fonction) dans un objet a besoin d'accéder à une autre méthode ou un attribut (variable) de ce même objet, il peut y avoir accès facilement grâce à "**this**" :

JS

```
let car = {

    speed: 50,

    readSpeed: function(){

        // "this" correspond en fait à l'objet dans lequel on est actuellement. this.speed veut donc dire "va chercher la valeur de l'attribut "speed" dans l'objet "car" "
        alert('La vitesse actuelle de la voiture est de ' + this.speed + ' km/h');
    }
};

// Affichera la phrase avec 50 dedans
car.readSpeed();
```

L’objet natif Math

L'objet Math est un objet natif dont les méthodes et attributs permettent de faire des maths plus avancées !

```js
// Retourne la valeur arrondie d'un nombre
Math.round(14.6)    // "15"

// Retourne un nombre à virgule pseudo-aléatoire compris entre 0 et 1 (1 exclu).
Math.random()       // "0.7497439643637321" (par exemple)
```

Voir plus de méthodes de l’objet Math : https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Math#méthodes

L’objet String et Number

L'objet String est un objet permettant de créer et représenter toutes les chaînes de texte utilisées en Javascript.

Quelques méthodes et attributs accessibles sur les chaînes de texte :

```js
// Retourne la taille de la chaîne
'chat'.length // "4"

// Retourne un caractère de la chaîne (premier caractère = 0)
'chat'[2] // "a"

// Permet de tester si la chaîne contient un mot (en réalité beaucoup plus que ça, match permet de tester des expressions régulières)
'le chat aime les arbres'.match(/chat/)    // true
```

L'objet Number est un objet permettant de créer et représenter tous les nombres utilisés en Javascript.

Quelques méthodes et attributs accessibles sur les nombres :

```js
let test = 50;

// Retourne le nombre arrondi à x nombre derrière la virgule (x = paramètre passé entre parenthèses)
test.toFixed(2) // "50.00"
```

Exercice 09

Énoncé :

Créer un Objet en Javascript qui sera :

Un animal

avec 4 attributs :

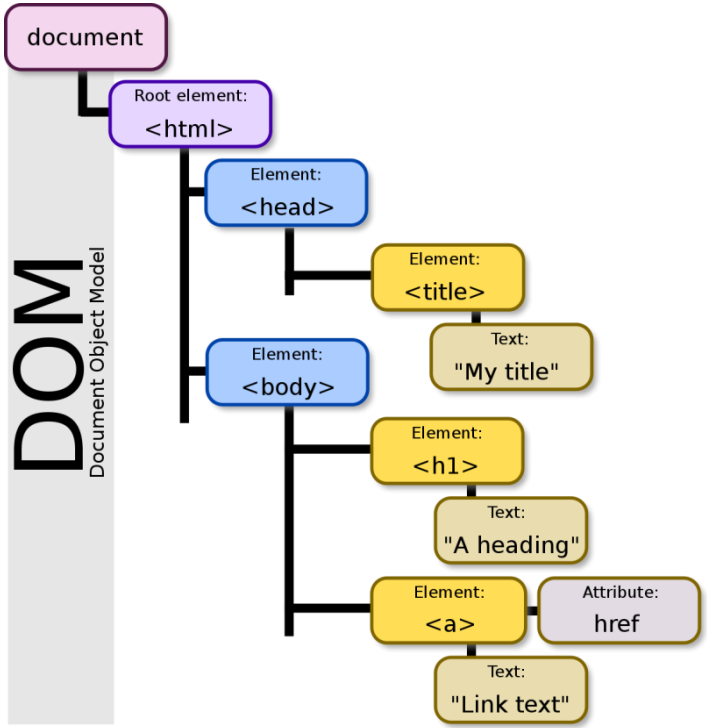
- un attribut sur le type d'animal (ex : chat, chien, loutre…),
- un attribut sur la provenance de l'animal (ex : Europe, Asie, Amérique…),
- un attribut sur le caractère de l'animal (ex: sociable, joueur, agressif, solitaire…)
- un attribut sur l'âge de l'animal (ex: 2, 10…)

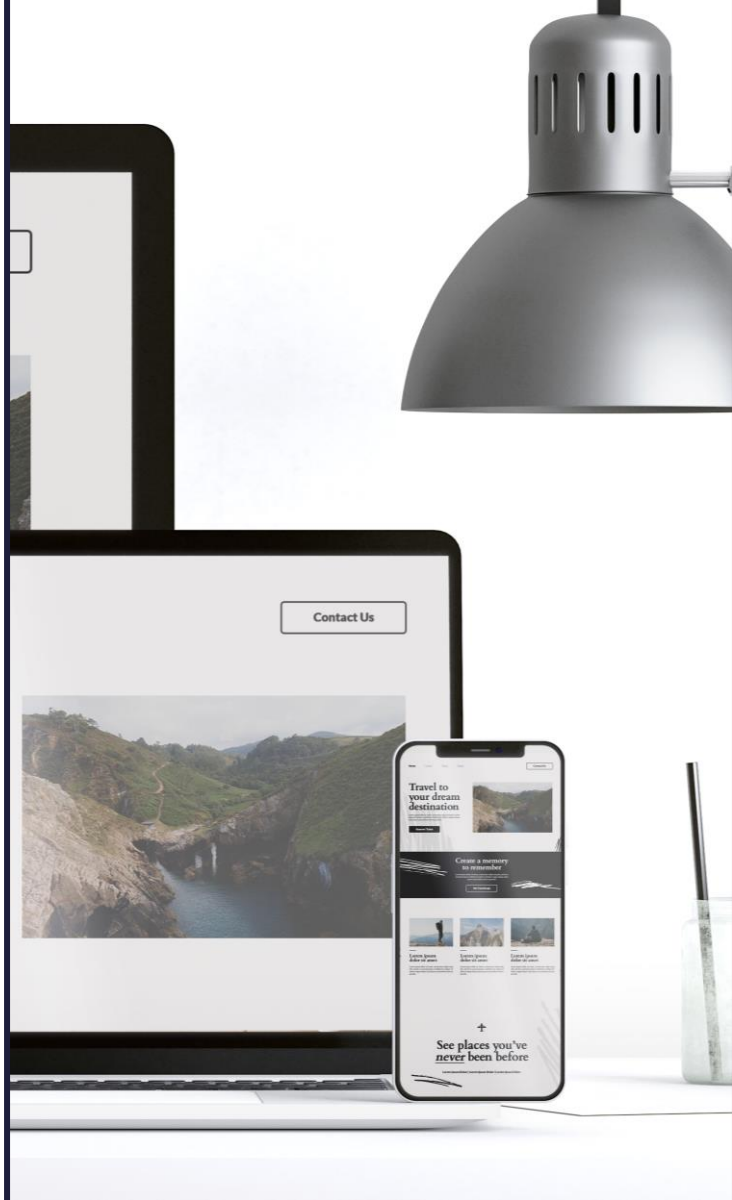
avec 2 méthodes :

- Une méthode manger qui produira un alert (« est entrain de manger ! »)
- Une méthode boire qui produira un alert (« est entrain de boire ! »)

Bonus : En dehors de l'objet, faire afficher avec une alert une phrase type qui reprendra les attributs : Il s'agit d'un chat originaire du Japon. Il a un caractère plutôt calme et il a 2 ans. Ensuite faire afficher dans une alert la méthode manger.

Manipulation du DOM





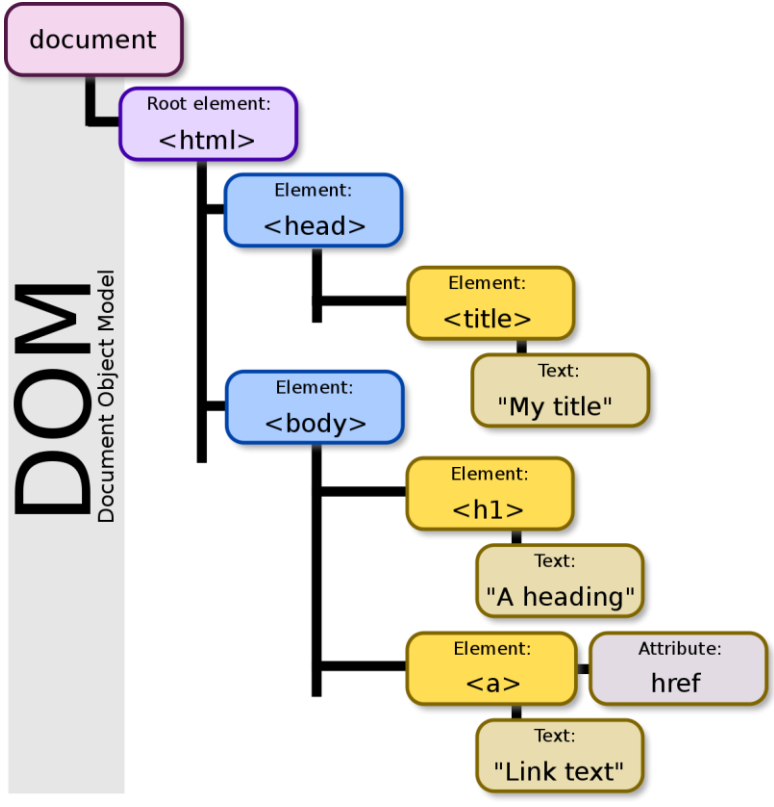
Le DOM

Qu’est ce que c’est ?

Le DOM (pour Document Object Model) est un objet Javascript créé par le navigateur qui représente l'intégralité de la page web actuelle. Il sert de point d'entrée pour que le développeur puisse agir sur la page web via Javascript.

Dans cet objet, toutes les balises HTML ainsi que tous les textes de la page web sont accessibles sous la forme de "nœuds". Javascript, à travers le DOM, nous permet de réaliser des actions directement avec ces nœuds :

- **Changer** le texte et/ou HTML d'un nœud
- **Modifier** les attributs HTML d'un nœud
- **Appliquer** du CSS sur un nœud
- **Dupliquer** un nœud
- **Supprimer** un nœud
- **Déplacer** un nœud
- **Créer** un nouveau nœud



Le DOM : Les sélecteurs 1/3

Pour pouvoir agir sur un nœud du DOM, il faut déjà le sélectionner. Il existe 5 sélecteurs permettant de sélectionner un ou plusieurs nœuds :

JS

```
// getElementById

// Sélection par un ID HTML (un seul élément peut être sélectionné au maximum)
let title = document.getElementById('main-title’);


// getElementsByClassName

// Sélection par une classe HTML (plusieurs éléments peuvent être sélectionnés : le
résultat sera sous forme d'un array dans tous les cas)
let redElements = document.getElementsByClassName('red’);


// getElementsByTagName

// Sélection par un type de balise HTML (plusieurs éléments peuvent être sélectionnés : le
résultat sera sous forme d'un array dans tous les cas)
let links = document.getElementsByTagName('a’);
```

Le DOM : Les sélecteurs 2/3

Les 3 premiers sélecteurs sont historiquement les plus anciens et aussi les moins utilisés, car remplacés par les deux derniers qui sont plus simples d'utilisation !

En résumé, avec querySelector on pourra sélectionner un seul élément, avec querySelectorAll plusieurs éléments (array). Les 3 autres peuvent être oubliés.

JS

```
// querySelector
// Sélection avec un sélecteur CSS (un seul élément peut être sélectionné au maximum)

// Ce sélecteur fonctionne exactement comme les sélecteurs CSS (on peut donc construire des sélecteurs très avancés comme en CSS)

let test1 = document.querySelector('#exemple');             // Sélection par ID
let test2 = document.querySelector('.red');                 // Sélection par classe (le premier élément de la page ayant cette classe)
let test3 = document.querySelector('strong');               // Sélection par balise (le premier strong de la page)
let test4 = document.querySelector('.main-navbar h1 a');    // Sélection plus avancée (le lien dans un titre h1 dans un élément ayant la classe ".main-navbar")


// querySelectorAll
// Sélection avec un sélecteur CSS (plusieurs éléments peuvent être sélectionnés : le résultat sera sous forme d'un array dans tous les cas)

// Ce sélecteur fonctionne exactement comme les sélecteurs CSS (on peut donc construire des sélecteurs très avancés comme en CSS)

let test1 = document.querySelectorAll('.red');                  // Sélection par classe (tous les éléments ayant cette classe dans la page)
let test2 = document.querySelectorAll('strong');                // Sélection par balise (tous les strong de la page)
let test3 = document.querySelectorAll('.main-navbar ul li a');  // Sélection plus avancée (tous les liens "a" dans un li dans un ul dans l'élément ayant la classe ".main-navbar")
```

Le DOM : Les sélecteurs 3/3

Les sélecteurs vus précédemment permettent aussi de sélectionner des éléments directement dans un autre :

HTML

```html
<div class="block">

    <h1>Lorem ipsum dolor.</h1>

</div>
```

JS

```js
// Sélection du bloc dans le DOM
let block = document.querySelector('.block');

// Selection du titre h1 directement dans le block
let title = block.querySelector('h1');
```

Parcourir l'arborescence du DOM 1/3

Une fois qu'un élément est sélectionné, on peut se déplacer dans l'arborescence à partir de cet élément :

HTML
```html
<div class="test">
    <h2>Lorem ipsum dolor.</h2>
    <h2>Architecto, dignissimos eius!</h2>
    <h2>Doloribus hic, laboriosam.</h2>
    <h2>Lorem consectetur adipisicing.</h2>
</div>
```

Sélectionner tous les enfants de l'élément :

JS
```js
// Sélection de l'élément
let testElement = document.querySelector('.test');

// Sélection de tous les enfants de l'élément (tous les h2)
let testElementChildren = testElement.children;
```

Sélectionner un seul enfant de l'élément :

JS
```js
// Sélection de l'élément
let testElement = document.querySelector('.test');

// Sélection du premier enfant de l'élément
let firstChild = testElement.firstElementChild;    // Premier h2

// Sélection du dernier enfant de l'élément
let lastChild = testElement.lastElementChild;      // Dernier h2

// Sélection du xème enfant de l'élément (celui que vous voulez, le premier = index 0)
let oneChild = testElement.children[2];            // 3ème h2
```

Parcourir l'arborescence du DOM 2/3

Sélectionner un élément frère de l'élément

HTML

```html
<ul>
    <li>Pomme</li>
    <li>Poire</li>
    <li class="target">Cerise</li>
    <li>Citron</li>
    <li>Kiwi</li>
</ul>
```

JS

```js
// Sélection de l'élément
let target = document.querySelector('.target');

// Sélection de l'élément frère situé juste avant l'élément
let previous = target.previousElementSibling;  // Sélectionne le li "Poire"

// Sélection de l'élément frère situé juste après l'élément
let next = target.nextElementSibling;  // Sélectionne le li "Citron"
```

Parcourir l'arborescence du DOM 3/3

Sélection du parent de l'élément

HTML

```html
<div>
    <p class="target">Exemple</p>
</div>
```

JS

```js
// Sélection de l'élément
let target = document.querySelector('.target');

// Sélection du parent de l'élément
let parent = target.parentElement;  // Sélectionne la div
```

Exercice 10

Énoncé :

1) Sélectionner l'élément ayant la classe ".start" dans le deuxième container

2) Sélectionner le 4ème "li" du premier container à partir de l'élément ".start", puis l'afficher dans la console.

Vous aurez à votre disposition les fichiers html et js (le fichier js sera à remplir).

1)

```
<div class="container">
    <h2>Sous-titre 2</h2>
    <div class="section">
        <h3>Sous-sous-titre</h3>
        <ul>
            <li>Lorem, ipsum.</li>
            <li class="start">Start !Commodi, ut!</li>
            <li>Necessitatibus, nostrum.</li>
        </ul>
    </div>
</div>
```

2)

```
<div class="container">
    <h2>Sous-titre 1</h2>
    <p>Lorem ipsum dolor sit amet.</p>
    <ul>
        <li>Lorem, ipsum.</li>
        <li>Commodi, ut!</li>
        <li>Cumque, aliquam.</li>
        <li>Je suis à sélectionner !</li>
        <li>Necessitatibus, nostrum.</li>
    </ul>
</div>

<div class="container">
    <h2>Sous-titre 2</h2>
    <div class="section">
        <h3>Sous-sous-titre</h3>
        <ul>
            <li>Lorem, ipsum.</li>
            <li class="start">Start !Commodi, ut!</li>
            <li>Necessitatibus, nostrum.</li>
        </ul>
    </div>
</div>
```

Manipuler le contenu textuel/HTML d'un élément 1/2

Récupérer ou modifier le contenu textuel d'un élément :

HTML

```html
<p class="target">Pomme</p>
```

JS

```js
// Sélection de l'élément
let target = document.querySelector('.target');


// Récupère le texte actuel contenu dans l'élément
let fruitName = target.textContent; // Récupère "Pomme"


// Modifie le texte contenu dans l'élément
target.textContent = 'Poire'; // Met "Poire" à la place de "Pomme"
```

Manipuler le contenu textuel/HTML d'un élément 2/2

Récupérer ou modifier le contenu HTML d'un élément :

HTML

```html
<p class="target">
    <strong>Pomme</strong>
</p>
```

JS

```js
// Sélection de l'élément
let target = document.querySelector('.target');


// Récupère le contenu HTML actuel dans l'élément
let elementContent = target.InnerHTML;  // Récupère "<strong>Pomme</strong>"


// Modifie le contenu HTML actuel dans l'élément
target.InnerHTML = '<i>Poire</i>';  // Met "<i>Poire</i>" à la place de
"<strong>Pomme</strong>"
```

Modifier les attributs HTML d'un élément 1/2

Javascript permet aussi de manipuler n'importe quel attribut HTML :

L'id HTML :

```js
// Sélection d'un élément
let target = document.querySelector('.target');

// Récupère l'ID HTML actuel de l'élément
let elementID = target.id;

// Change l'ID HTML actuel de l'élément
target.id = 'new-id';
```

Les classes HTML :

```js
// Sélection d'un élément
let target = document.querySelector('.target');

// Récupère la valeur entière de l'attribut "class" de l'élément
let elementClasses = target.className;

// Change l'attribut "class" de l'élément
target.className = 'red main';
```

Modifier les attributs HTML d'un élément 2/2

Les classes HTML :

JS

```js
// Retire une classe CSS sans toucher les autres
target.classList.remove('red');

// Ajoute une classe CSS sans toucher les autres
target.classList.add('blue');

// Remplace une classe CSS par une autre sans toucher les autres
target.classList.replace('blue', 'red');

// Ajoute une classe CSS si l'élément ne l'a pas, sinon retire la classe s'il l'a déjà
target.classList.toggle('green');

// Test si l'élément contiens une classe CSS ou pas
if( target.classList.contains('green') ){

    alert("L'élément contient bien la classe CSS green");
} else {

    alert("L'élément ne contiens pas la classe CSS green");
}
```

Les autres attributs HTML :

JS

```js
// Sélection d'un élément
let target = document.querySelector('.target');

// Récupère le contenu d'un attribut de l'élément par son nom (href, alt, src, title, type, value, etc...)
let attributeValue = target.getAttribute('src');

// Modifie un attribut de l'élément par son nom
target.setAttribute('alt', 'Photo de chat');
```

Appliquer du CSS sur un élément et supprimer un élément

S'il est possible de changer les propriétés d'un élément HTML en jouant avec ses classes, on peut aussi changer directement des attributs CSS dessus :

```js
// Sélection d'un élément
let target = document.querySelector('.target');

// Change la propriété CSS "color" de l'élément
target.style.color = 'red';

// Attention, les propriétés CSS composées de plusieurs mots doivent être écrites en lower camel case
target.style.fontSize = '4rem';
```

Pour supprimer un élément, la technique est un peu déroutante au départ. Il faut sélectionner le parent de l'élément pour ensuite supprimer son enfant :

```js
// Sélection de l'élément à supprimer
let targetToDelete = document.querySelector('.target');

// Suppression de l'élément en passant par son parent
targetToDelete.parentElement.removeChild( targetToDelete );
```

Déplacer un élément

Il existe 4 déplacements d'éléments HTML fondamentaux avec JS :

- **before** : Déplacer un élément avant un autre
- **after** : Déplacer un élément après un autre
- **prepend** : Déplacer un élément dans un autre au début avant ses autres enfants
- **append** : Déplacer un élément dans un autre à la fin après ses autres enfants

HTML

```html
<div class="block1">
    <p class="target">Lorem ipsum dolor.</p>
</div>

<div class="block2">

</div>
```

JS

```js
// Sélection de l'élément à déplacer
let targetToMove = document.querySelector('.target');

// Sélection de l'élément qui servira de référence au déplacement (on va déplacer .target dans .block2)
let destination = document.querySelector('.block2');

// Déplacement de "target" dans "destination"
destination.append( targetToMove );
```

Créer et dupliquer un élément

Pour créer un élément, il faut d'abord le former puis ensuite l'insérer dans le DOM avec after / before / append / prepend :

HTML
```html
<div class="block1">

</div>
```

JS
```js
// Création d'un nouvel élément de type paragraphe (p)
let newElement = document.createElement('p');

// On donne un contenu textuel à notre élément
newElement.textContent = 'lorem ipsum dolor';

// Pour le moment cet élément n'existe pas dans la page web, on l'insère donc dans l'élément .block1
document.querySelector('.block1').append( newElement );
```

Dupliquer un élément se fait assez facilement :

JS
```js
// Sélection de l'élément à dupliquer
let target = document.querySelector('.target');

// Création d'une copie de l'élément qui sera stockée dans la variable (true = copier aussi les enfants de l'élément)
let copy = target.cloneNode(true);

// Maintenant ont fait ce qu'on veut de la copie (modifications, insertion dans le DOM, etc...)
```

Exercice 11

Énoncé :

1) Créer une fonction JS permettant de créer un overlay HTML et de l'insérer dans la page (dans le body)

2) Créer une fonction JS permettant de supprimer l'overlay sur la page

Vous aurez à votre disposition les fichiers html et js (le fichier js sera à remplir).

Avant de se lancer dans l'exercice, on va d'abord créer l'overlay nous même en html/css.

Agir sur un array d'éléments

Quand on utilise un sélecteur qui retourne un array de plusieurs éléments (comme querySelectorAll), on ne peut pas directement manipuler tous les éléments, il faut les parcourir avec une boucle :

HTML

```html
<ul>
    <li>Lorem ipsum dolor.</li>
    <li>Libero, quae, quia!</li>
    <li>Facere, quos, vitae!</li>
    <li>Reprehenderit similique, voluptatum.</li>
    <li>Officiis, repellat soluta!</li>
</ul>
```

JS

```js
// Sélection de tous les "li" dans le "ul"
let elements = document.querySelectorAll('ul>li');

// Si on veut mettre tous les éléments de l'array en rouge, il faut d'abord parcourir tous le tableau pour ensuite appliquer la transformation sur chaque élément un par un
elements.forEach((element) => {

    // mise en rouge de chaque élément de l'array (un par tour)
    element.style.color= 'red';

});
```

Agir sur des champs de formulaire

Pour récupérer la valeur d'un champ de formulaire :

HTML
```html
<input type="text" class="target">
```

JS
```js
// Sélection du champ de formulaire
let field = document.querySelectorAll('.target');


// Récupération du texte qui est actuellement dans le champ
let fieldValue = field.value;
```

Les datasets

Les datasets sont des attributs HTML spéciaux (qui commencent par "data-") qui permettent de stocker des informations directement dans le code HTML (un peu comme des variables dans le code HTML)

Ces attributs spéciaux sont valides au W3C et peuvent avoir n'importe quel nom du moment qu'ils commencent par "data-" :

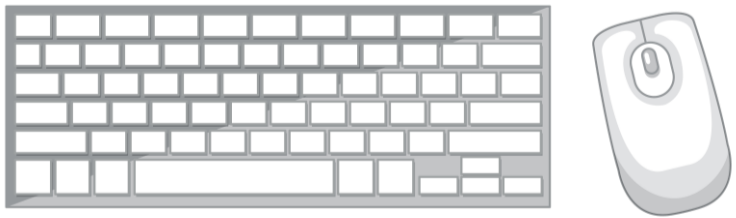
HTML

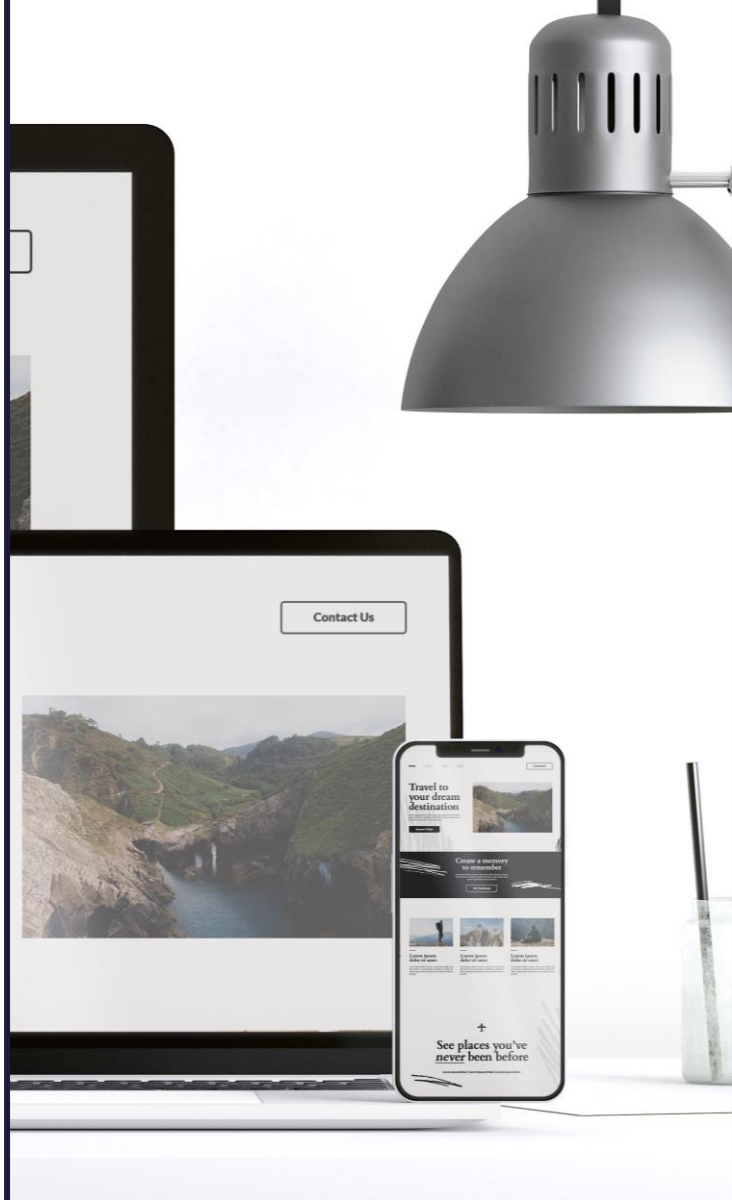
```html
<!-- Cette balise contient un dataset contenant un nom de ville -->
<h2 data-city="paris">Jean</h2>
```

JS

```js
// Récupère "paris"
let cityName = document.querySelector('h2').dataset.city;
```

Écouteur d’évènement





Les écouteurs d’évènements

Qu’est ce que c’est ?

Un écouteur d'évènement est une fonction rattachée à un élément HTML dans le DOM. Cette fonction sera exécutée à chaque fois qu'un évènement sera déclenché depuis l'élément HTML (click, passage de souris, etc...).

HTML

```html
<h1>Lorem ipsum dolor sit.</h1>

<h2>Libero quibusdam recusandae repudiandae.</h2>
```

JS

```js
// Mise en place d'un écouteur d'évènement au "click" de souris qui sera rattaché au titre h1
document.querySelector('h1').addEventListener('click', function(){

    // Le titre h2 change de couleur en rouge
    document.querySelector('h2').style.color = 'red';

});
```

Types d’écouteurs d’évènements

Liste non exhaustive d'écouteurs d'évènements JS :

- **click** : se déclenche au clique gauche de souris
- **dblclick** : se déclenche au double clique gauche de souris
- **mouseenter** : se déclenche quand la souris entre sur l'élément
- **mouseleave** : se déclenche quand la souris quitte l'élément
- **mousemove** : se déclenche quand la souris se déplace sur l'élément
- **keydown** : se déclenche quand une touche du clavier s'enfonce
- **keyup** : se déclenche quand une touche du clavier remonte
- **focus** : se déclenche quand un élément récupère le focus clavier
- **change** : se déclenche quand un champ de formulaire change de valeur
- **submit** : se déclenche quand un formulaire est envoyé
- **reset** : se déclenche quand un formulaire est reset

Liste complète des évènements ici : https://developer.mozilla.org/fr/docs/Web/Events#listing_des_événements

This

Si depuis l'intérieur d'un écouteur d'évènement on souhaite utiliser l'élément qui a déclenché l'événement, on peut y a accéder facilement avec le mot-clé "**this**".

HTML

```html
<button>Clique-moi pour me changer de couleur !</button>
```

JS

```js
// Mise en place d'un écouteur d'évènement au "click" de souris qui sera rattaché au bouton
document.querySelector('button').addEventListener('click', function(){

    // this = l'élément qui à déclenché l'écouteur d'évènement, donc le bouton dans cet exemple
    this.style.color = 'red';

});
```

Exercice 12

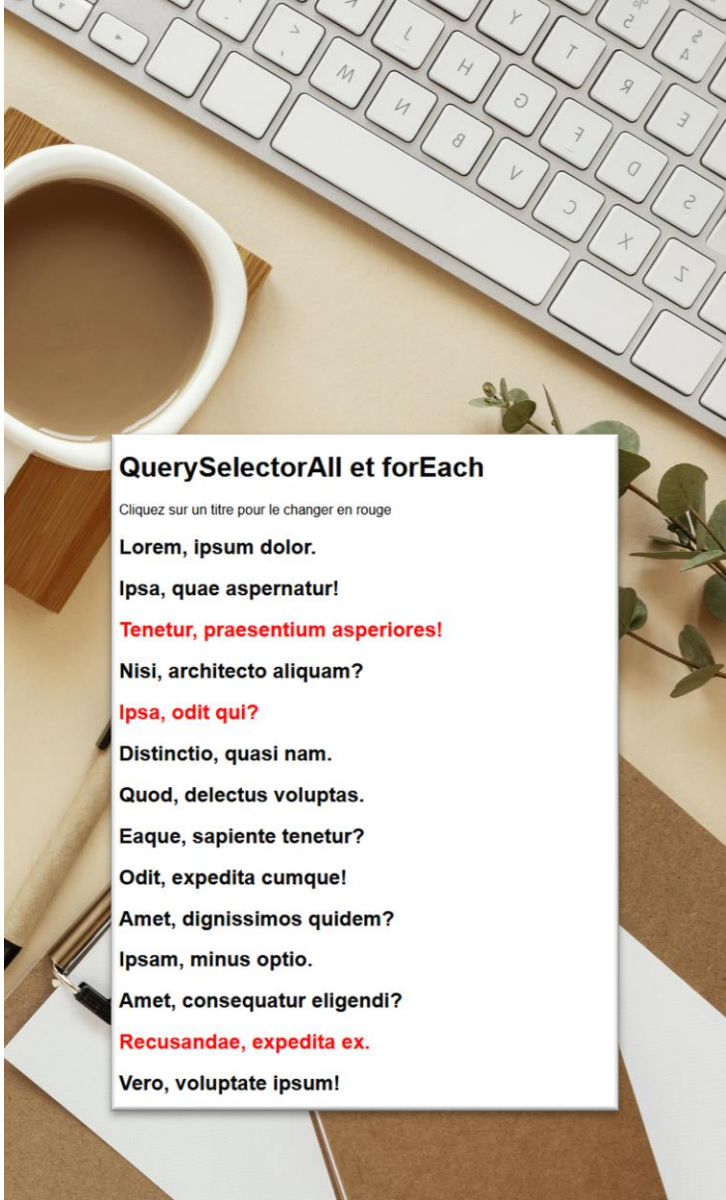
Énoncé : Plein de titres en rouge !

Le fichier HTML sera fournit !

- Le but est de reproduire la même chose que sur cette image dans laquelle, lorsque **l'on clique sur un des titres** listés, le titre cliqué devient rouge.

- Chaque titre cliqué doit changer en couleur rouge.

Bonus : si on reclique sur un titre, il revient en noir !



Empêcher le comportement par défaut d'un élément HTML

Depuis un écouteur d'évènement, il est possible de stopper le comportement par défaut d'un élément HTML (comme les liens ou les formulaires)

HTML

```html
<a href="https://www.google.fr">Lien vers Google</a>
```

JS

```js
// Dans cet exemple, le lien ne fonctionnera jamais
document.querySelector('a').addEventListener('click', function(e){
    e.preventDefault();
});
```

Attention à ne pas oublier de déclarer le paramètre "e" pour avoir accès à la méthode preventDefault dans la fonction.

- En général bloquer le comportement par défaut d'un élément HTML est utile quand on souhaite par exemple faire des vérifications sur les champs d'un formulaire sans que ce dernier ne recharge la page.

Supprimer un écouteur d'évènement déjà mis en place

Il existe 2 méthodes pour supprimer un écouteur d'évènement déjà mis en place sur un élément HTML :

- Soit on supprime directement l'élément HTML sur lequel est positionné l'écouteur (radical, ça supprime tout)
- Soit on supprime juste l'écouteur d'évènement (mais pour ça il faut que la fonction de l'écouteur d'évènement soit non anonyme) :

JS

```js
// Fonction de l'écouteur d'évènement
function test(){
    alert();
}

// Mise en place de l'écouteur d'évènement
document.querySelector('.exemple').addEventListener('click', test);


// Suppression de l'écouteur d'évènement
document.querySelector('.exemple').removeEventListener('click', test);
```

Attendre le chargement complet de DOM

Si on essaye de faire des actions sur des éléments HTML qui ne sont pas encore chargé par le navigateur (si par exemple le fichier JS est inclus dans le head par exemple), ça ne fonctionnera pas. Il faut retarder l'exécution du code JS pour attendre que tout le DOM ait fini de charger.

JS

```js
// Écouteur d'évènement un peu particulier, directement rattaché au DOM
document.addEventListener('DOMContentLoaded', function(){

    // Code JS qui sera exécuté après chargement complet du DOM...

});
```

Fin du module

par Audrey Donjon

m2iformation.fr